High-Risk Technologies
Applications to Fusion (and Accelerator) Physics

Presenter: J.R. Cary†*

†Tech-X Corporation; *University of Colorado

October 7, 2008

Session Chair: Phil Colella

Preparatory remarks
Risks for application developers
Rewards in fusion and accelerators
Promising technologies
Horizons: general remarks
Some project ideas

Office of Science

TECH

Step 4: Catalog your transgressions: I am an outlier

Palahniuk (Choke): The fourth step in the twelve-step process is to keep a record of your addiction, recording all your transgressions, past and present.

- C++ for scientific computing since 95
- As soon as I heard about it, I had
  - Sound on my group web page
  - A nonlinear dynamics applet
- Re fusion: really an outlier
- I am an OASCR sympathizer
- But my bread and butter depends on doing good computational science

Who should drive tool development? Toolers (tool developers) or appsers (application developers)?

- Technology driven products can miss the mark
  - Excessive development cycle (debug, build, test)
  - Five-step workflows do not require complex management
  - Oversimplifications (distinguishing services on basis of interface)
- But Market driven products are often too late in a small field like computational science
  - Computational scientists have to come up with a solution to survive, so already have I/O, component model, grids, data structures
- Decoupled tool/apps development best practices:
  - Toolers respond to need, provide something better
  - Toolers provide migration path
  - Appsers adopt on next code cycle (e.g., modernization program)
- Tooler opportunities best in fields that are in a modernization program or have a history of modernizing

Collaboration is the right approach to developing useful tools

- See the needs of application developers first hand
  - Fast I/O: don't give up metadata
- See the drivers of application developers first hand
  - Science! Not FLOPS, abstraction, …
- Working with apps developers give appreciation for hurdles (Why are those appsers so stupid as to not see the advantages of my technology?)
- One approach
  - Build application
  - Extract good ideas into libraries
  - Refactor application to use libraries

For software reuse, this latter is our approach at Tech-X

Risks: why would an application developer adopt new tools, *even if paid*?

The web is littered with approaches

- **Numerical Java**
- **HPC, HPFortran**
- **Haskell, APL, ….**
- **The Grid?**
- **SCONS**
- **POOMA**

The computational tool developer makes his career on ideas, even if the tool is not used

The computational scientist risks his career by making the wrong tool choice

What have apps developers learned about new languages, promises of Nirvana, …?

Types of Risk

The point of these projects is to develop new capabilities that, from the point of view of the applications community, are too high-risk to be carried out as business as usual. Classify the risks of possible projects in the following categories:

o Well-characterized application of a new technology – risk comes from using an exotic methodology in a production science application, requiring hardened implementations of the methodology and a bridge between the apps domain and the experts in the new technology. Example: implementing an existing model in a new programming language or programming framework.

o Well-established technologies applied to a new problem area – risk comes from whether the methodology can be successfully modified to meet problem-specific needs. Example: AMR for climate.

o Fundamental new approaches, particularly in domains where there is little prior art in modeling. Example: Heterogeneous spatial modeling in cell biology.

o Other risks not listed here.

Computational application developers prefer the bottom of the software risk hierarchy

- Failure (e.g., technology dropped by sponsor) requires project do-over
  - Language change
  - Giving up main
  - Adopting another's data structures
- Failure requires extensive code rewriting
  - Code to fundamentally different approach (messaging -> threading)
- Failure requires minimal code rewriting.
  - Code to new API
- Failure requires switching to a new executable
  - Replace GnuPlotPy with matplotlib
- Failure requires switching out people

Basili et al, "Understanding the High-Performance Computing Community: A Software Engineer's Perspective," IEEE Software, Jul/Aug, 29 (2008).

Shasharina et al, "FACETS – A Multiphysics Parallel Component Framework," COMPFRAME 2008 (Karlsruhe, Germany).

Scientific programmers often develop code such that they can plug in different technologies to evaluate them. For example, when MPI was new in the 1990s, many groups were cautious about its long-term prospects and added it to their code alongside existing message-passing libraries. As MPI became widely used and trusted, these older libraries were retired. Similar patterns have been observed with solver libraries, I/O libraries, and tracing tools.

The languages being developed in the DARPA HPCS program were intended to extend the frontiers of what's possible in today's machines. So, we sought practitioners working on very large codes running on very large machines. Because of the time they've already invested in their codes and their need for long-lived codes, they all expressed great trepidation at the prospect of rewriting a code in a new language.

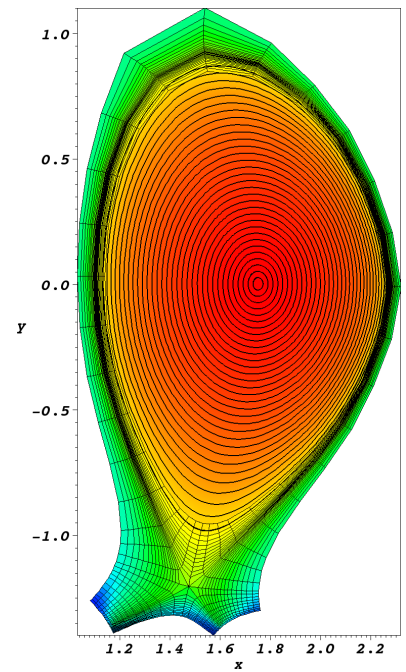
Conclusion: A new technology that can coexist with older ones has a greater chance of success than one requiring complete buy-in at the beginning.

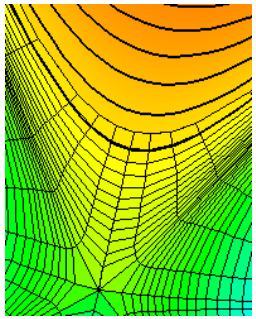
Fusion applications: to predict catastrophic macroscopic motion or turbulent transport

- **Core**
  - **Macroscopic motions, but started by boundary layer instabilities**
  - **Microturbulence determines transport fluxes**
  - **Fast particles**
    - **Neutral beams**
    - **RF generated**
    - **Fusion products**
- **Edge**
  - **Macroscopic motions, ELMs**
  - **Moderate turbulence**
  - **Chaotic field lines**
  - **Atomic physics**
  - **Wall interaction**

We got particles!

Impact requires ALL of the physics





Rewards stem from scientific discovery (from there, general codes)

General	Fusion	Accelerators
• Discovery of a new mechanism • Predict the parameters of the dominant mechanism • Outline a series of experiments to determine the dominant mechanism	• Neoclassical tearing mode stabilization and/or drive to unstable; ELM generation by peeling-ballooning. • Normalized beta as indicator for disruptions • Show pedestal height depends on some parameter combination	• Self trapping in LWFA; crab cavities • Which cavity has the greatest gradient without quenching? What pulse shape is best for self trapping? • Compute optimal cavity, experiment does final adjustment

Technologies that look promising (to me)

Multigrid, AMG, other solvers routine, but I keep my datastructures. Now decisions being made on developer convenience

(flexible build, true serial, …)

- **Autotuning (really needs embedded CS/Amers = toolers)**
- **Massively parallel on chip (GPU, Cell) for both fields or particles**
- **Higher-order embedded boundaries (how can we librarify?)**
- **Divergence preserving ADI electromagnetics (too specialized?)**

more maturation needed

- **Component concepts could use development within applications, reluctant to give up main**
- **New languages: must be C(C++)-callable, usable in existing frameworks**

more risky

- **Load balancing libraries: do I have to give up my data structures?**

minimally risky

- **Visualization, including ray tracing**

Horizon: general remarks

- For each project, answer the following questions.
  - o Give a timeline for progress, over a 3-5 year time scale.
  - o What would be an optimal end state in 10 years ?
  - o What is the level of effort required to meet these goals ?
  - o What would be the organizational structure of the collaboration between the apps and the high-risk math / CS team ?
  - o What external dependencies need to be taken into account (e.g. existence of supporting SciDAC infrastructure)?

- **Any project has to produce physics within two years.**
- **After 10 years, a production code for both government and industry**
- **Central, interdisciplinary team of 3-5 with contributors from other institutions.**
- **Central team with both appsers and toolers, additional remote teams with well-defined tasks**
- **Local development environment.**

Project 1: Autotuned PIC codes

- PIC codes routinely get better performance with sorting
  - Particles in cell [0, 0], then [0, 1], then …
  - Timing tells when to sort
  - But particles gather fields from 4 cells
  - Better to sort all in cells {[0, 0], [0, 1], [1, 0], [1, 1]} followed by next 4 and so on? Group by 9?
- Particles routinely grouped
  - Particles come and go in a simulation
  - Memory management aided by use of *ragged arrays* of <u>some maximum size</u>

<u>RISKS</u>

- Invasive coding may not be modularizable

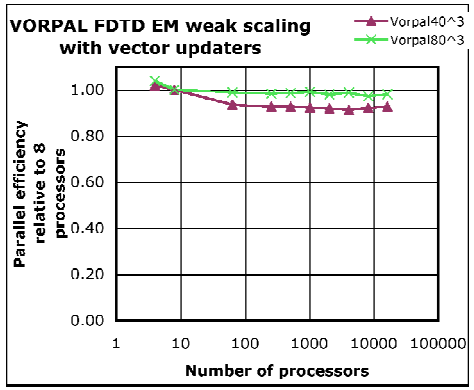
<u>REWARDS</u>

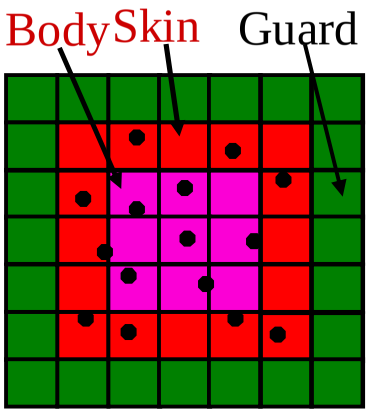
- Productivity increases by factors of 10 on existing hardware

Autotuning could help us figure our the best parameters for these and other coding choices

Project 2: an autotuned multi-mini-domained electromagnetic PIC code

- Very small domains parallelize well with FDTD on largest hardware
- If on same chip, can these domains be even smaller? Smaller enough to fit in cache?
- Two-level indexing?





RISKS

- Invasive coding may not be modularizable

REWARDS

- Productivity increases by factors of 10's on modest multicore

Project 3: GPU or Cell based computations of multi-mini-domained systems

- **See previous slide**

RISKS

- **Invasive coding may not be modularizable**

REWARDS

- **Productivity increases by factors of 100's on massive multicore**

Migration path:
Project 1, 2, 3

Project 4: Higher-order embedded boundary computations

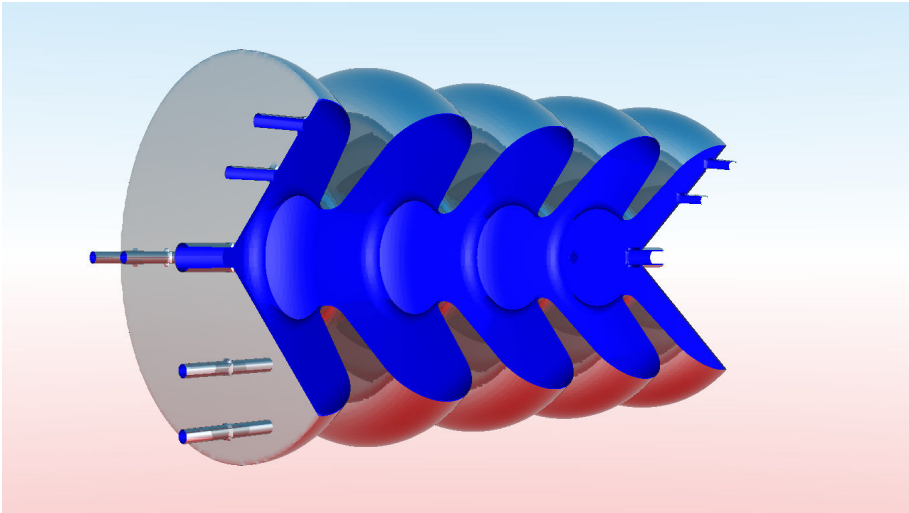
- Embedded boundary algorithms provide blazing speed with accuracy
- How do we create a library?

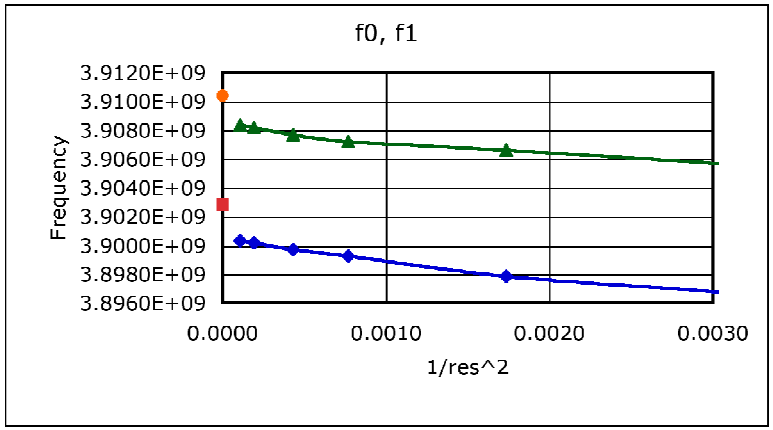
RISKS

- Asymmetric matrices give long-time instabilities

REWARDS

- Increased accuracy, speed





Project 4: GPU or cell based computations of EM using ADI

- ADI approaches have natural data breakup
- Always stable
- Transpositions?

RISKS

- Generalization to higher order problematic?

REWARDS

- Absolutely stable, time steps determined by problem not numerics

Project 5: Co-development of concurrent parallel component frameworks

Unabashedly FACETS oriented

- **Component in broadest sense means only modularity, so motherhood, apple pie, …**
- **What does it mean in physics?**
  - **Interfaces, but there is something more: two components meeting at a surface provide the same interface, but**
  - **Implementations: simple models to complex**
  - **Instances: multiple neutral beams**
- **What does it mean in CS?**
  - **Runtime discovery**
- **High risk project**
  - **extract, general software from an application framework**
  - **build an application in a new area**

Projects 6-∞: More and more risk

- **ASCR load balancing software? (Give up data structures?)**
- **Multiblock fluid solver for edge plasma? (Needs better reduced models in the edg.)**

RISKS

- **Giving up data structures…**

REWARDS

- **Faster convergence**

Need more time to flesh out

No conclusions

this is a workshop!